Acnet Data Requests/Settings

System Implementation

Thu, Sep 15, 1994

Introduction

The message formats for Acnet data requests/settings is described in the Acnet Design Note 22.28. It uses the Acnet header designed by Charlie Briegel to support generalized task-task communications across a network. The Network Layer software in the VME Local Stations supports these Acnet header-based messages. This note describes the implementation of the support for Acnet Data Services data acquisition and setting messages.

Message flow

When a request or setting message is received, it is directed to a well-known taskname RETDAT for requests and SETDAT for settings. (These 6-character network task names are encoded in the "Radix-50" form used by PDP-11 and Vax computers.) At initialization, the Acnet Request Task creates a message queue (called ACRQ) that is used to receive Acnet header-based messages directed to the taskname RETDAT or to the taskname SETDAT. NetCnct registers both tasknames to the Network Layer. (By directing both message types to the same queue, processing of the messages in original network order is assured. One can issue a setting command and immediately issue a request to read back the setting value and still be confident of obtaining the new setting, assuming a valid setting.)

```
Function NetCnct (taskName, queueId, eventMask, VAR taskId);
```

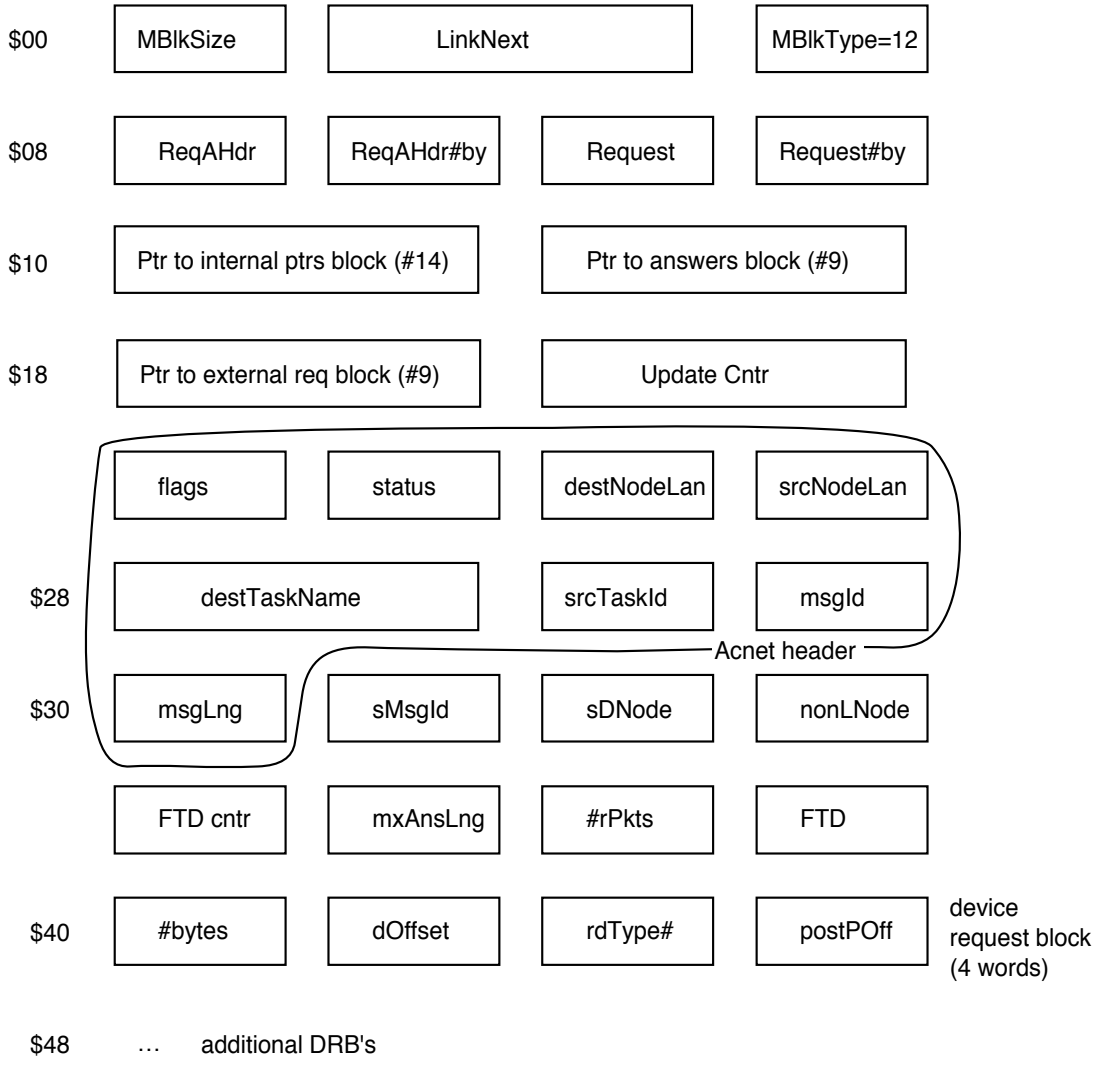
The eventMask is left zero, as the Request Task will simply wait on the message queue rather than wait on an event. The Request Task then enters an infinite loop that calls NetCheck to wait for a message and, upon receiving one, processes it.

```
Function NetCheck (taskId, timeOut, VAR msgRef);
```

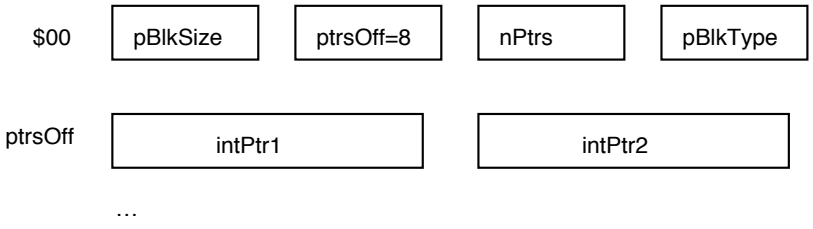
When the function returns with valid status, the message type is checked as found in the first word of the Acnet header. If it is a USM (unsolicited message) with the CAN (cancel) bit set that was directed to RETDAT, the request identified by the message id is cancelled. If it is a USM that was directed to SETDAT, the setting message is processed immediately with no acknowledgment message.

If the message type is a request, the message following the header is checked. If it is a setting, it is processed immediately, and an acknowledgment is returned in the form of a status-only reply message (Acnet header only). If it is a request for data, then 3 message blocks are allocated for support of the new request. (If the request specifies an existing active message id, then the existing request is cancelled.) The basic request block (type#12) houses the various parameters needed to monitor the request activity. Two pointers are included in that block that point to the other related allocated blocks—the internal ptrs block (type#14) and the answers block (type#9).

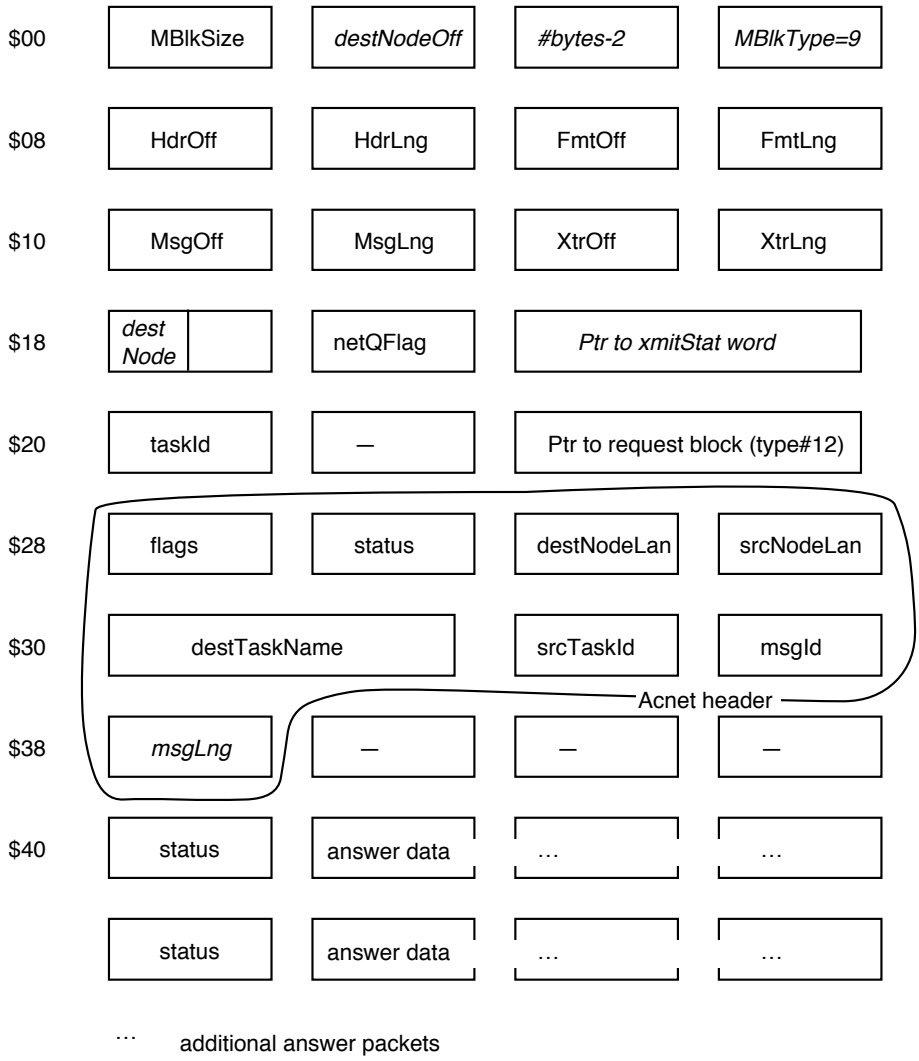
The basic Acnet request block (type #12) contains the array of device request blocks (DRB's) and the frequency time descriptor (FTD).



The Internal Ptrs block (type #14) contains the array of internal ptrs that are used to update the request (build the answers) efficiently.



The answers block (type #9) is an Acnet message block of the form used by the Network Layer software when the answers are to be returned to the requesting node/task. It also includes a pointer to the parent request block (type #12) for use by QMonitor for one-shot requests that need automatic cancellation.



After the request support blocks have been filled, the basic request block is inserted into the chain of active data requests using `INSCHAIN`. It is inserted at a position adjacent to another request block made by the same node, if any, in order to increase the likelihood of combining the answer responses of multiple requests into the same network frames. Then the Update Task is triggered to update the request and build the first set of answers immediately.

The request message is processed as it resides in the network frame input buffer `DMA`'d into memory by the chipset. This processing includes "compiling" the request into the `DRB`'s and the internal ptrs array for later update processing. The message count word in the network frame buffer is decremented to signal to the network that the request message space is now free for future use. Note that initializing the request as it resides in the network buffer (instead of using `NetRecv` to copy it into the caller's buffer) saves copying the ident arrays in the request, at the expense of the additional responsibility of decrementing the message count word when finished with the request message.

Updating requests

The Update Task scans through all active requests each cycle to update any that are due for processing. It checks for this new request block type (#12) and builds the answers accordingly. The read-type routines are called for each listype using the array of internal pointers to build the answer data.

When the Update Task has built answers that are to be returned to the requester, it invokes the `NetQueue` routine to do it. Just before that, however, it calls `NetXChk` to flush any existing queued messages that are going to a different node or use a different protocol type (different `SAP`) to the network chipset. This is to ensure prompt delivery of responses to different nodes and yet combine answer messages directed to the same node into the same frame for greater network efficiency.

```
Function NetXChk (newNode, newType): Integer;

Function NetQueue (taskId, VAR msgBlk, VAR xmitStat): Integer;
```

The Update Task flushes all queued messages to the network after it has processed all active requests each 15 Hz cycle.

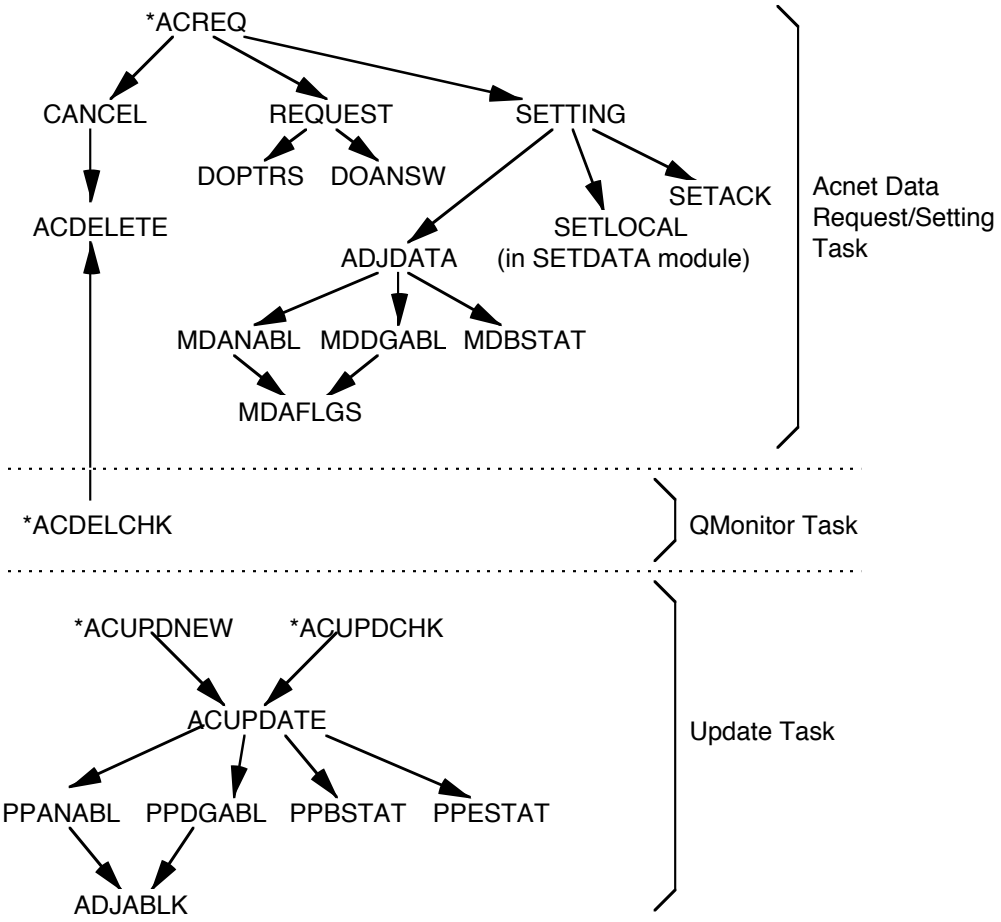
Acnet Settings

Processing setting messages, as compared with data requests, is greatly simplified because it is all done immediately and no dynamic data structures need be prepared for later update processing. The destination task name of `SETDAT` indicates that the message is a setting.

The many set-type routines have been enhanced so that they now return error codes whenever they encounter errors. (Previously, the setting was simply ignored.) This error response word is used in the setting acknowledgment message. A zero status indicates no detected error in performing the setting. This acknowledgment is returned only if the setting message type is a request. If the setting message is a `USM`, no acknowledgment is returned.

Acnet Request Module Road Map

The organization of the routines in the ACREQ module is as follows, where an asterisk denotes a declared entry point:



The upper collection of routines comprise the Acnet Data Request Task, which waits for a message directed to the destination taskname RETDAT or SETDAT and processes it. For a *request* message, the CANCEL routine searches the active list chain for a match against the message id ("list#"), the requesting node and source task id. If it finds a match, it calls ACDELETE to cancel that active request. The REQUEST is the bulk of the code which prepares the request block, internal pointers block and answers block for later processing by the Update Task. It uses several other routines to help break that job down into more manageable pieces.

For a *setting* message, the setting action is performed immediately. The system routine SETLOCAL is called to process each packet. An error return aborts the processing of any remaining settings in the message, and SETACK is invoked to deliver the setting acknowledgment status-only reply message.

The middle section is the ACDELCHK routine which is called by the QMonitor Task when it has detected the completion of transmission of an Acnet-type message (block type#9) with bit#6 and bit#5 of the NetQFlg word set in the block, indicating that the block is to be retained for re-use and that it is a *Acnet* protocol request as opposed to a

DZero protocol request. It checks for the case of a one-shot Acnet data request that should be cancelled. So QMonitor has to recognize the type#9 message and be aware of the `NetQFlg` word. It also looks for the case of the type# `0xF9` and frees the memory of that block. (A type# `0xF9` block is an altered type#9 block no longer needed for holding an Acnet answer response but could not be freed when cancelling the Acnet request because bit#7 of the `NetQFlg` was set indicating that the block was queued for transmission to the network.)

The last section includes two entry points that are called by the Update Task to process type#12 requests during its traversal of the chain of active requests. `ACUPDNEW` updates the request only if it has never been updated before, whereas `ACUPDCHK` examines the FTD counter and updates the request only if it is due. `ACUPDATE` shepherds the actual updating of the request and queues an answer response to the network.

Error reporting for requests

A number of potential errors are detected when processing an Acnet data request message. For most of these, a response is returned to the requester consisting of a status-only reply, which includes only the Acnet header. Current error codes are as follows:

-32 spare
-33 invalid message size
-34 spare
-35 invalid #request packets
-36 dynamic memory unavailable
-37 invalid listype#
-38 invalid identype (error in listype table)
-39 invalid ident length for listype#
-40 invalid #bytes requested per ident
-41 invalid total #idents this request
-42 size of answers too large
-43 size of answers > max length given
-44 nonzero data offset not supported in request packet
-45 nonzero data offset not supported in setting packet
-46 invalid #setting packets
-47 invalid read routine type# (error in listype table)
-48 node# does not match this system's node#
-49 invalid destination task name

In addition to the response to the requester, these errors are recorded in the Local Station in local variables of the Acnet Request Task. They can be inspected for diagnostic value (with suitable instruction). For each error, a data word is recorded for the last error of that type followed by a count word of the number of errors of that type that have occurred since the station was reset.

Another error that can be returned by the Network Layer itself is the following:

-21 destination task not connected to network (`RETDAT` or `SETDAT`)

This means that the 4-byte destination task name in the Acnet header was not recognized by the node that received it. For systems which have Network Layer

support but have not yet been updated with the Acnet data request software, this will certainly result.

Setting acknowledgment error codes

The following list of errors can occur in response to a data setting message:

0	No error. Setting successful.
-65	System table not defined for this listype.
-66	Entry# (chan#, bit#, etc) out of range.
-67	Odd #bytes of data
-68	Bus error
-69	#bytes too small
-70	#bytes too large
-71	Invalid #bytes
-72	Set-type out of range (error in listype table)
-73	Settings not allowed for this listype
-74	Analog control type# out of range (error in analog descriptor)
-75	Invalid binary byte address in BADDR table
-76	Invalid mpx channel# (Linac D/A hardware)
-77	F3 scale factor out of range (motor #steps processing)
-78	No CPROQ table or co-proc# out of range
-79	Hardware D/A board address odd
-80	Bit# index out of range (associated bit control via channel)
-81	Bit# out of range for this system's database
-82	Digital Control Delay table full (for software-formed pulses)
-83	Digital control type# out of range 1–15
-84	Co-processor command queue unavailable
-85	Co-processor invalid queue header
-86	Queue full or unavailable
-87	Dynamic memory allocation failed
-88	Error status from 1553 controller
-89	Invalid 1553 command for one word output
-90	Invalid 1553 Command Block address (must be multiple of 16)
-91	Invalid 1553 order code in first word of Command Block
-92	1553 interrupts not working
-93	Cannot initialize 1553 command queue
-94	No Q1553 table of pointers to 1553 controller queues
-95	Invalid Motor table
-96	Motor table full
-97	Invalid 9513 timing channel pair
-98	Timing event# out of range.
-99	Invalid data value.
-100	Invalid #bytes of text in Comment alarm control
-101	No DSTRM table of Data Stream queue pointers
-102	Data Stream queue type# out of range
-103	Data Stream queue not initialized
-104	No MMAPS table of memory-mapped board templates
-105	Invalid MMAPS table header
-106	Invalid MMAPS table entry size

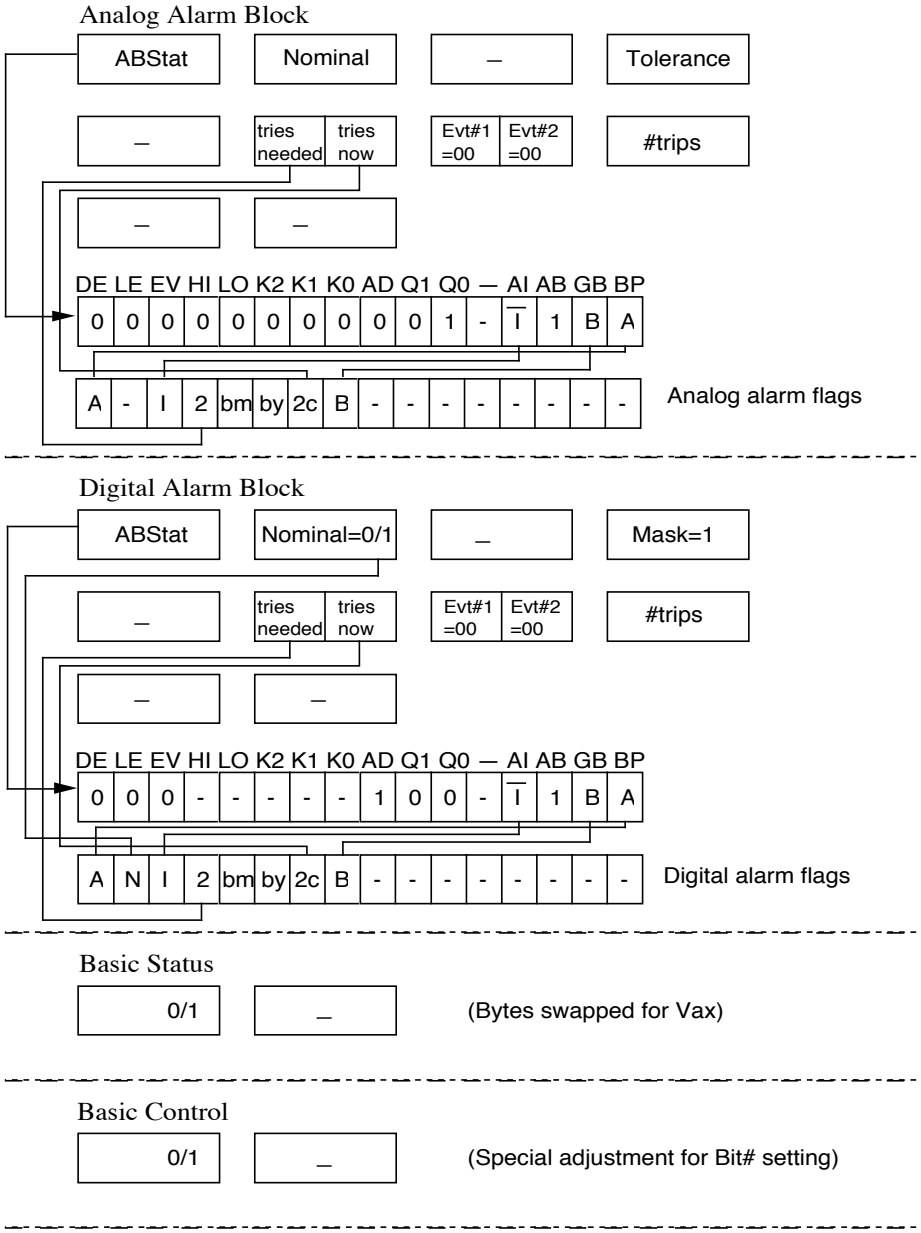
-107 Invalid board# for `MMAPS` table
-108 Invalid directory entry in `MMAPS` table
-109 End of `MMAPS` table reached during template processing
-110 Invalid `MMAPS` command type code
-111 Invalid `MMAPS` loop params
-112 Invalid `MMAPS` nested loop
-113 spare
-114 Invalid listype#
-115 Invalid ident type# (error in listype table)
-116 Invalid ident length for this listype
-117 Little console settings switch disabled
-118 Little console external settings switch disabled
-119 Data Server setting not implemented
-120 Invalid listype for this Acnet property

Data format conversions

Special considerations of the Acnet protocol require support of several standard data formats. Logic is included that supports the following standard record structures:

`ANALBL`	Analog Alarm Block
`DGALBL`	Digital Alarm Block
`BSTATS`	Basic Status
`BCNTRL`	Basic Control
`ESTATS`	Extended Status

These standard data formats are as follows:



The alarm blocks are the most complex structure to support. The flag word must be edited to conform to the Acnet standard form in response to a data request. And it must be edited to the Local Station format in response to a setting. The other fields are

similarly edited. The tries needed byte may be one or two, according to the 2x bit in the analog or binary alarm flags. The #trips word is returned as extra info in the alarm block. Event-related alarms are not supported.

A special adjustment must be made to accommodate data requests of less than 6 bytes for an analog alarm block. When the read-type routine is invoked to update the answers to such a request, the #bytes requested must be set to at least 6, or the read-type routine will not return the analog alarm flags word that must be edited to make up the `ABStat` word in the reply. This adjustment also requires that 6 extra bytes be allocated in the answers block (type#9) in order to assure that the extra bytes requested of the read routine cannot be written beyond the end of the block.

For the case of the Basic Status property, the bytes of answers must be swapped to conform to the byte order of the DEC machines. This is also true of some forms of Basic Control, but the data sent with listype #21 (digital I/O via Bit#) is considered a word, where the hi byte is the digital control type# and the lo byte is the pulse delay (when used). So in this case, the bytes should not be swapped.

Limitations of present implementation

Features *not* supported in the initial version of Acnet request handling are the following:

SSDR-related requests
Event-style FTD's
Data offset

It is not intended to support data requests of the "Data Server" type for the Acnet protocol. Idents in a request are *ignored* if they do not include the node# of the local station receiving the request in the first word of the ident. This means that one could send the same request to a group of nodes using the functional group multicast form of network addressing, and each node receiving the request would select out its own idents for answer response. (Obviously the requesting node would need to scan the original request in order to be able to match the answers with the questions.) Currently, however, the Acnet header-based protocols do not permit sending request messages to a group of nodes.

Comparison with "classic" protocol

The Acnet `RETDAT/SETDAT` protocol for data requests/settings is a very flexible protocol that serves multiple front end computers whose internal software may be organized quite differently. The SSDN component of the request/setting packets is the key that makes it work. The coding of the 8-byte SSDN structure can be designed for the needs of each front end; neither the console computers nor the central database cares what it is. It only must be correctly entered into the central database.

The "classic" protocol that has been used by the Local Station processors since 1982 is designed to support that particular front end type. The concept of characterizing data requests in terms of *arrays of idents* to be processed in the same manner is used to optimize request update efficiency. Updating an array of channels with analog readings, for example, is distilled down to a 3 instruction loop with the loop count being the number of channels in the array.

This implementation of the `RETDAT/SETDAT` protocols does *not* rearrange the request into one that can be processed optimally. It can be enhanced at a later date if the extra effort is deemed to be worth the increase in efficiency.